INTERFACE

IDL

The Interface Description Language for Star Wars Online needs to be constructed so that the User Interface can be built from script (either from the client side or server side). The IDL handles the display characteristics of the interface, as well as the behaviors of interface elements in response to user input. It is used to define the location, art, hue, id, behavior and other characteristics of menu elements. It also handles the default behaviors of interface elements on events (such as mouse click). Ideally, a layout tool will be created to dump all of this information into a data file or script format so that a developer can plug it into code on the client or server.

Vision

- It should have an intuitive user interface. If the choice for initial interface is between flexibility and ease of use, we will choose ease of use, and make the flexibility an option.
- The game must be built to be reusable even if no such plans exist; this will ensure that Live support will be easier.
- A high level of coding standards will be adhered to, so that split development runs into a minimum of problems.
- It must be easy to extend and add content to the game after ship.
- An interface description language will allow easy addition of new interfaces.

Terms

Widget: The smallest window element that can be interacted with. Usually a button or other control.

System details

Notes

These are fairly raw notes, not design requirements. Commentary is welcome of course ☺.

- Some events don't need to be verified by the menu creator (server or client). For example, moving through tabs doesn't need to be verified…
- Need to support keyboard repeat
- Need to support aliases
- GUI Tool Possible commercial tools: Flash, Shockwave, Director

Example of what the layout tool might do: A designer wants to add a new pop-up dialogue box when the player talks to a particular NPC. They use the Layout tool create a menu that includes a textbox, ok button, and cancel button. S/he then edits the properties of each widget and assigns it an ID. All of this information is then dumped out into a data file or script file in the syntax of the IDL so that it can be added to code on either the server or client.

- Features: Skins, color schemes, loading bit maps, moving interface elements?
- Windows need to be able to communicate with other windows.
- Window Manager. Window desktop?

Widgets

Interface element. There is a C++ class for each widget and a script equivalent to trap the events. This is just a list of possible widgets:

1. Text Box – wish list: being able to select text out of the text box. Hyperlinks – could be done with button down and close.
2. List Box
3. Menu Bar
4. Radio Button
5. Bitmap
6. Static text
7. Edit box
8. Group box (buttons)
9. Check Box
10. Spin Controls
11. Drop Box
12. Slider Box
13. Progress Bar
14. Hot Keys
15. Lists
16. Trees
17. Tabs
18. Animation
19. Time Picker
20. Calendar
21. Extended Combo box
22. Color Picker
23. Selection Wheel
24. Context Sensitive/Tear Out Menus
25. Dials
26. Knobs
27. Sliders
28. Interactive map
29. Draw
30. Multiplayer chalkboard: (a wishlist item). A map with a second layer for drawing. Ability to save off info, and then load backup and continue to modify.
31. Scroll Bar
32. File Browser
33. Splitter
34. Status bar
35. HTML window
36. List control (multiple columns and sorting)
37. Tool bar
38. Drag and Drop
39. Time line?
40. 3d Render Window

Client side events

(a) Kb
    1. down
    2. up
    3. click

(b) Mouse
    1. button click
    2. button dclick
    3. button down
    4. button up
    5. wheel up/down
    6. move (move within boundaries of widget, timer)
    7. idle

(c) Timer

Joystick should map to keyboard and mouse events.

Likely Custom Controls
Ship customization
PDA –nav map
Mission Selection – space and ground maps? 3d?

CAMERAS AND CONTROLS

The purpose of this document is to outline the basic camera views available in the game, as well as to describe the kinds of control that players will have over them. This document does not attempt to define the interface for these controls. At this time, it is assumed that the game can be played from both 1st and 3rd person perspectives.

Vision

- We must impress the user within 30 seconds via dramatic visuals, impressive use of sound and music, and compelling interaction.
- The game should offer a sense of discovery.
- The play environment should be customizable so that the players can change it to suit.
- A high degree of customization for both players and ships is a key requirement.
- Big visual payoffs are important.
- There must be a sense of exploration.

Terms

Avatar (player/character/ship): The physical representation of the player in the game world.

Camera (camera view): The image shown to the player.

Occlusion: Obstruction or Blockage of the camera's view of its subject.

System

Details

All the camera views in this document will fall into one of three categories: First Person, Third Person, and Remote. You will also notice that these there is a surprising amount of overlap between space and ground when it comes to camera control. As a matter of fact, the only significant difference is that on the ground, there is always a source of occlusion that has to be dealt with in the 3rd person/Remote: the terrain. This is not so for space.

There are a LOT of sub-variants of third person cameras and quite a few different control schemes for each. The camera views and their controls will always apply on both ground and space unless otherwise noted.

Each camera view will have the following elements defined: Name, Purpose, Physical Orientation, Movement, Occlusion, and Additional Details.

Camera Views

1st Person Move

Purpose: The 1st Person Camera View is a general use camera. It allows the player to view the world as if they were actually there, and see it through the eyes of their avatar.

Physical Orientation: The camera's position is located at the head of the avatar, looking through the face (or in the case of space, it is located in the cockpit of the craft looking forward).

Controls: The player can look in all directions. The avatar's entire body will twist and turn so that the range of motion is not limited. The camera moves in concert with the movement and orientation of the avatar's entire body (lead by the head and upper torso (depending on the animation system)). Likewise, in space, the camera moves with the cockpit.

Occlusion: The player's should not occlude by his/her own avatar with the exception of an appendage or an equipped item.

Additional Details: N/A

1st Person Look

Purpose: The 1st Person Camera Look View is used when the player wishes to look around in first person without changing their entire body's physical orientation. It allows the player to view the world as if they were actually there, and seeing it through the eyes of their avatar.

Physical Orientation: The Camera's position located at the head of the avatar, looking through the face (or in the case of space, it is located in the cockpit of the craft looking forward). When the player uses this camera in a space ship, they look around the inside of the cockpit.

Controls: Player's range of movement is constrained by the avatar's head/torso animation. The player can only look as far to the left, right, up or down as the avatar can animate. In space, the camera moves inside the cockpit within the constraints of the avatar's head movement.

Occlusion: The player should not be occluded by his/her own avatar with the exception of an appendage or an equipped item.

Additional Details: N/A

3rd Person Free Lock

Purpose: The 3rd Person Free Lock View is an excellent camera for the player to examine themselves and their local whereabouts. This view allows the player to rotate and zoom the camera around their avatar.

Physical Orientation: The camera is located outside the body of the avatar, but always faces it. When the player moves, the camera continues to point in the same direction at all times (if the camera was facing south to north and the player rotates clockwise, the camera does not rotate with the player).

Controls: The player may zoom the camera view in very close or zoom out a great distance (so close that the avatar takes up more than the entire view and so far that the avatar takes up less than 10% of the screen). The player may rotate the camera around the player and change it's angle of view. The camera will collide with terrain and this will be the limiting factor on viewing the avatar from the lower hemisphere (if they find a cliff, they can look up their own pants leg I suppose…)

Occlusion: Occlusion does matter in this view. Please see the section labeled occlusion to see possible solutions for occlusion in 3rd person cameras.

Additional Details: This camera type has can be used for a few default camera views including the ever popular "Birds eye" view from above. The UO point of view can be created with this camera type.

3rd Person Fixed Lock

Purpose: The 3rd Person Fixed Lock View differs from the free lock because it's orientation rotates with the avatar.

Physical Orientation: The camera is located outside the body of the avatar, but always faces it. When the player rotates, the camera rotates with the player.

Controls: The player may zoom the camera view in very close or zoom out a great distance (so close that the avatar takes up more than the entire view and so far that the avatar takes up less than 10% of the screen). The player may rotate the camera around the player and change it's angle of view. The camera will collide with terrain and this will be the limiting factor on viewing the avatar from the lower hemisphere.

Occlusion: Occlusion does matter in this view. Please see the section labeled occlusion to see possible solutions for occlusion in 3rd person cameras.

Additional Details: Several default camera views could be created with this type: Traditional Tomb Raider "ass cam", A close chase cam (above and behind the head), etc…

Remote Cameras

Purpose: Remote camera views are used to give the player a view of some part of the game that is physically removed from the avatar. Examples of remote cameras are: Security cameras, Video Feeds, Guided weapons, remotely controlled druids, etc…

Remote camera views may or may not take up the full view of the player's normal camera view. For example, a player may receive remote camera view from his or her droid that appears in a small square in the corner of their normal view.

Physical Orientation: Depends on the specific purpose of the remote camera.

Controls: Depends on the specific use of the camera.

Occlusion: see occlusion section.

Occlusion Solutions

There are several possible courses of action to deal with occlusion and it is possible that some combination of them will provide the ideal solution. The prime concern of any solution to this problem is to keep the player immersed and oriented to their position in the world at all times.

Clipping

The simplest solution to the problem is to not draw the offending object (polygon section/bitmap) that is blocking the view. While fast and easy, this solution has several drawbacks. First, it raises the possibility of allowing a player to see areas they aren't meant to.

For instance, if the player wants to see what's in a locked room, all they'd need to do is put their camera in 3rd Person Fixed Lock, move the camera behind their head, and then back up until the camera passes into the room. The system strips away the polygons of the wall, but the objects in the room that are not obstructing the view to the player are drawn. A possible solution to this phenomenon is to avoid drawing items that the player's avatar cannot see. Unfortunately, LOS calculations are not computationally inexpensive and would have to reside on the server. Another solution is to not care if the player can see things in this manner.

Another general problem with this solution is that it makes it difficult for the player to keep situational awareness about their physical surroundings. In the example above, the player wouldn't be able to visually see where the wall exist, and therefore might not know that they are bumping into one.

Transparency

A similar solution to clipping, it has the same drawbacks except that it allows the player to keep better situational awareness because they can see physical boundaries.

An additional thought for both clipping and transparency is that when a poly is clipped or removed, if it is a child object, the rest of the object should be clipped or removed. This would prevent the "cookie cutter" effect and would also make it possible for the player see objects that might otherwise be occluded.

Automatic Camera Movement (smart cam)

ACM refers to the idea that a camera system could adjust its position to keep the player in view. Many game systems use some form of smart camera like this.

Some camera systems merely zoom the camera in and out when occlusion occurs. In the example above, as the avatar backed up towards the wall, the camera would begin to move closer to the player once it came into contact with the wall. There are two drawbacks to this solution. First, as the player backs up, either the camera ends up zoomed in so close that the avatar fills the field of view, or it prevents the avatar from backing up once the minimum range of the camera is reached. If there is a minimum range, it becomes a challenge for the player to move around in tight spaces. The second drawback is that when the avatar is in tight spaces, the camera is constantly zooming in and out. This behavior can be jarring to the player. See Ultima Ascension's first level when the player must move around in a house. Hallways are particularly nausea inducing...

Combining changes to camera distance with some other solutions might provide better results. For example, perhaps once the camera moves in, it would move out at a much slower pace, minimizing the constant in and out motion. Another change is to make the player transparent as the camera gets closer. This would minimize the problem where the player obstructs the view.

Other camera systems actually move the camera laterally and vertically to find a better view of the player. The major drawback to this problem is that the player tends to have difficulty keeping their orientation within the world. As the camera moves around the player navigation gets increasingly difficult, especially if the player is trying to keep up with other entities in the world (like combat). Several platform games use this solution (see Mario64 as an example).

Additional Features

Third Person Scene (combat/conversation)

Purpose: This is a cinematic camera view used when the player has an obvious target of interest (either through combat, conversation, or otherwise).

Physical Orientation: The camera positions itself so that 2 or more targets can be seen at the same time.

Controls: The camera should attempt to change angles depending on who spoke last, or took the most interesting action. For example, when used in a conversation, the camera might check every 5 seconds to see who spoke last. Once that person is determined, it changes position so that the speaker is in the foreground and the recipient(s) are in the background.

Occlusion: see occlusion section.

PHYSICS

Star Wars Online will make use of two different types of physics systems in order to present the game world in a realistic way while at the same time saving on system resources by not doing unnecessarily complicated computations for situations that don't require them.

System Details

Two Systems will be used

1) Inverse Kinematics
2) Force - Based

For Star Wars Online, we are currently talking about using two different physics systems for different situations. One of these physics systems would be of the IK or Inverse Kinematics type, meaning that it only worries about the velocities that items are traveling at or trying to attain. The other physic system to be used is the force type, which concentrates on what forces are reacting upon an object, and these forces end up making objects behave in a particular way.

Advantages and Disadvantages of each system

1) System Resource Use
2) Fidelity

Each system has its own advantages and disadvantages. The primary advantage of the IK system is that it is less demanding of system resources than a force-based physics system. The primary advantage of the force-based physics system is that the movement produced by it tends to look more natural than movement produced by the other less complicated physics systems.

The more demanding but higher-fidelity force-based physics system will be used only for thrust-based vehicles, giving their movement and behavior a more realistic look and feel. Thrust-based vehicles include types like speeder bikes, land-speeders, submarines, and snow-speeders. For any vehicles that travel while in contact with the ground such as carts, Sand-Crawlers, AT-AT's or mounted animals, the lower-fidelity, less-demanding IK physics system is used.

Additional Resource Conservation Methods

Objects at rest ignored for physics calculation purposes

As an additional savings to system resources, physics calculations will not be applied to objects that are at a state of rest. This helps keep the game from bogging down when you consider the incredible number of objects that will be lying around all over the game world. Your standard crate lying on the ground is a good example of this kind of item. Furthermore, resting items can be classified as being one of three types. One type includes objects that don't collide, such as an object that you might step over. Another type would be any immobile object, such as a fire hydrant, which is anchored to the

ground. The last type would be an object that can be collided with and is then moved by the collision, such as a soccer ball. Any resting object that is affected by a collision would have its resulting behavior resolved using the Inverse Kinematics physics system, until it is again at rest. As the player comes up and 'kicks' the soccer ball, the ball immediately begins checks against the IK physics system until it resolves the collision, and re-attains its resting position.

Case-Based System Details

People and vehicles in contact with the ground

Vehicles in contact with the ground would be less apt to slide around, so driving a Sand Crawler or an AT-AT would use the simple IK physics system to go whatever direction you drove it, until you let off of the 'go' button. At that point, some sort of minimal pseudo inertia would bring the vehicle to a halt almost immediately. Once at rest, all physics calculations would stop until it once again began moving, whether under it's own power or as the result of some sort of impact or accident.

For characters that are simply walking around on the ground, we are only concerned that they go where the player wants them to go, when they want them to go there. Since they are walking around on the ground, a straight IK system would do quite nicely. We don't expect the person to be skidding in the corners or leaning into turns or anything, so the fidelity of a force-based system is not called for. If our player-avatar was running over the ground and then ran up a hill and jumped off it, then game would let him move at his max speed over flat terrain, would then reduce his speed as he went up the hill based on the general average local slope of the hill, and would handle bringing him back to a resting position on the ground after leaving its surface. The instant the player let off of the 'go-forward' button, the player would lurch to a halt.

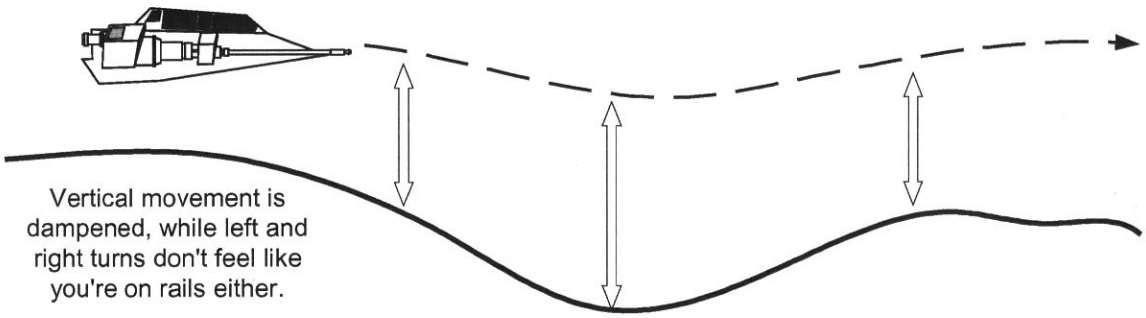
For a player driving a Sand-Crawler around, the player would simply drive it around like a car, but because it is using the IK system, would not ever attempt to skid in the corners, lean into turns. However, because it is a massive item, it would not immediately stop, but would rather rapidly lurch to a halt (a slight discrepancy).

Chart 1. One example of an Inverse Kinematics application.

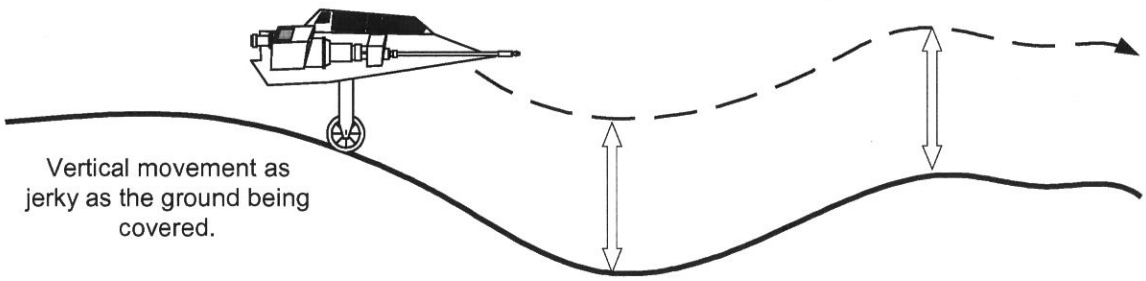
Flying/Gliding/Hovering Vehicles

Although we aren't planning to allow fully unlimited flight over the ground, we do intend to allow 'hovering' vehicles to glide over the ground and using the force-based physics system for them will allow for some of the most accurate-looking movement possible. Since having a floating vehicle simply stuck on the end of a stilt would give fairly jerky terrain-following behavior, the employment of the force-based system will smooth the craft's flight-path as it glides over the terrain's high-points, and will allow for more natural turns and swerves as well.

Chart 2. Clarification of the usage of force-based versus Inverse Kinematics in a 'flight' application.



Not This:



Objects will tend to return to a 'resting' condition

Any vehicle that leaves its resting position will have IK physics applied to it until it re-attains its resting position. This includes hovering and skimming vehicles that aren't currently flying at their selected 'float' altitude. Another example would be a person that has jumped off a cliff and uses the IK physics system to bring it to a 'resting' position at the foot of the cliff.

Physics in space

Spacecraft in space would use the force physics system. When a person wants to accelerate, the ship has a forward moment of force applied to the ship and this accelerates it. The only role that velocity plays in the game is as a speed limit. As a spacecraft tears through a turn at high speed, it would skid a variable amount in the direction of the outside of the turn depending on the ship's weight, its 'maneuverability' attributes, perhaps the pilots' skill, and the skid would limited by whatever maximums we saw fit to include. Also, a fully laden freighter would accelerate and decelerate more slowly than its empty brethren.

The force dynamics would also be used in the combat system. If a ship is hit in its side by a missile, it might actually skid a bit opposite the direction of the impact. A ship with a very powerful tractor-beam could drag a small, low-powered vessel backwards or be able to hold it in position.

®, ™, and © 2000 by Lucasfilm Ltd. All Rights Reserved.

Star Wars Galaxies Design Document property & copyright 2000 Sony Online Entertainment

Design document written by
Anthony Castoro
Nick Newhard
Reece Thornton

With assistance by
J. Allen Brack
Raph Koster
Ryan Palacio

Screenshots assembled by too many people to list.

Concept art by Arnie Jorgensen

Special thanks to Haden Blackman, Kevin McCann, and Danny Han